



Visualizing and representing the evolution of topological features

Fonseca, Rasmus; Jørgensen, Desirée Malene Schreyer

Publication date:
2012

Document version
Early version, also known as pre-print

Document license:
[Unspecified](#)

Citation for published version (APA):
Fonseca, R., & Jørgensen, D. M. S. (2012). *Visualizing and representing the evolution of topological features*. Abstract from Computational Geometry: Young Researchers Forum, Chapel Hill, United States.

Visualizing and representing the evolution of topological features

Rasmus Fonseca*

Desirée Malene Schreyer Jørgensen†

Abstract

Simplicial complexes are discrete representations of topological spaces that are practical for computational studies. The first three Betti-numbers (indicating the number of components, tunnels and voids), as well as the topological persistence of each such feature, is well-defined and can be efficiently computed for simplicial complexes embedded in 2D and 3D [1, 2].

We introduce a novel representation of the evolution of topological features in simplicial complexes using so-called tunnel-trees in 2D and void-trees in 3D. This new representation makes it possible to analyze topological evolution by applying tools for analysis of binary trees. Furthermore it supplies a new method for visualizing topological evolution.

Introduction

A simplicial complex, \mathcal{K} , is a set of simplices where any face of a simplex in \mathcal{K} is also in \mathcal{K} and the intersection of two simplices in \mathcal{K} is either empty or a face of both simplices. Delfinado and Edelsbrunner [1] define a *filter* to be a sequence of simplices, $\sigma^1, \sigma^2, \dots, \sigma^n$, where $\mathcal{K}_i = \{\sigma^1, \sigma^2, \dots, \sigma^i\}$ is a simplicial complex for any choice of i (see left part of Figure 1). The filter represents the evolution of a simplicial complex and will be the focus of the methods described here. The topological features of a complex can be described using the Betti-numbers, β_d , which indicate the rank of the d th homology group. The first three Betti-numbers ($\beta_0, \beta_1, \beta_2$) can be interpreted more intuitively as the number of components, holes, and voids respectively. A $\mathcal{O}(n\alpha(n))$ -time algorithm exists to calculate the evolution of β_d as a simplicial complex is grown using a filter [1]. This method identifies each k -simplex, σ^i , as either *positive* if it creates a new k -cycle and thereby increases β_k , or *negative* if it changes a k -cycle into a k -boundary and thereby decreases β_{k-1} . For each positive k -simplex, σ^i , the negative $(k+1)$ -simplex, σ^j , that is responsible for turning the k -cycle, created by σ^i , into a k -boundary can be efficiently identified [2]. The difference between the indices of such two simplices is defined to be the *persistence* of the k -cycle represented by σ^i .

*Department of Computer Science, University of Copenhagen, rfonseca@diku.dk

†Department of Computer Science, University of Copenhagen, daisy@diku.dk

Tunnel- and void-trees

One interesting observation about tunnels in simplicial complexes embedded in 2D is that, often, when a positive 1-simplex (edge) is added to the complex, it splits one tunnel in two. If the empty space around the complex is considered a *bounding tunnel*, then every positive edge will split an existing tunnel in two. Similarly, if the entire space around a simplicial complex embedded in 3D is considered a *bounding void*, then a positive 2-simplex (triangle) always splits an existing void in two.

Based on this observation we define a *tunnel-tree* (or β_1 -tree) of a 2D filter to be a binary tree where each node represents a distinct tunnel (see right part of Figure 1). The root is the bounding tunnel, and the leaves are triangular tunnels that will not be split further. With each node n we associate the positive edge that represents the tunnel, $\epsilon(n)$, and with each leaf, we associate the negative triangle that fills this tunnel, $\tau(n)$. The tunnel-tree is ordered such that for any node n , the triangle of the rightmost leaf, $\tau(\text{TREE-MAX}(n))$, is the triangle that 'destroys' $\epsilon(n)$ and hence determines its persistence. A *void-tree* (or β_2 -tree) of a 3D filter is defined in a similar fashion, only with positive triangles as nodes and negative tetrahedra as leaves.

A β_k -tree is constructed by running through the filter backwards as shown in Algorithm 1. Leaves are created when a negative $(k+1)$ -simplex is encountered and the roots of leaves are connected when positive k -simplices are encountered.

Algorithm 1 Build a β_k -tree given a filter

```

1: Create a 'bounding node',  $n_b$ 
2: for  $i = n$  to 1 do
3:   if  $\sigma^i$  is a negative  $(k+1)$ -simplex then
4:     Create a new node,  $n$ , and set  $\tau(n) \leftarrow \sigma^i$ 
5:   else if  $\sigma^i$  is a positive  $k$ -simplex then
6:      $(n_0, n_1) \leftarrow$  Nodes of the two  $(k+1)$ -simplices
       adjacent to  $\sigma^i$ 
7:      $(n_0, n_1) \leftarrow (\text{ROOT}(n_0), \text{ROOT}(n_1))$ 
8:     Swap  $n_0$  and  $n_1$  if  $\tau(\text{TREE-MAX}(n_0))$  is
       younger than  $\tau(\text{TREE-MAX}(n_1))$ 
9:     Create a new node  $n$  with  $n.\text{left} \leftarrow n_0$ ,
        $n.\text{right} \leftarrow n_1$ , and  $\epsilon(n.\text{left}) \leftarrow \sigma^i$ 
10:  end if
11: end for
12: return  $\text{ROOT}(n_b)$ 
```
